hadoop
APACHE
HBASE

Apache HBASE

Agenda

- ✓ What is HBase?
- ✓ HBase Features & Use-cases
- ✓ HBase Run Modes
- ✓ Getting started with HBase
- ✓ HBase Data Model
- ✓ HBase Commands
- ✓ HBase Architecture

What is HBase?

- NoSQL Database (Non-SQL, Non-Relational)
- Distributed column-family oriented database
- Built on top of HDFS (HDFS as storage layer)
- Interacts via HBase shell or it's Java API
- Runs on a Cluster of machines (linearly scalable)

HBase Features

- Highly Scalable

Slaves nodes can be added on the fly to increase scalability.

HBase Features

- Highly Scalable
- Dynamic Schema

Columns can be added on the fly. Columns are not part of the table schema.

HBase Features

- Highly Scalable
- Dynamic Schema
- Distributed Database

Data gets partitioned automatically as the data grows. HBase provides “automatic sharding” - tables are dynamically distributed by the system to different region servers when they become too large.

HBase Features

- Highly Scalable
- Dynamic Schema
- Distributed Database
- Native to Hadoop

Can utilize MR framework and can use HDFS for storage there by gets all the advantages of Hadoop.

HBase Use Cases

Real time analytics

Search Engines

Storing RSS feeds

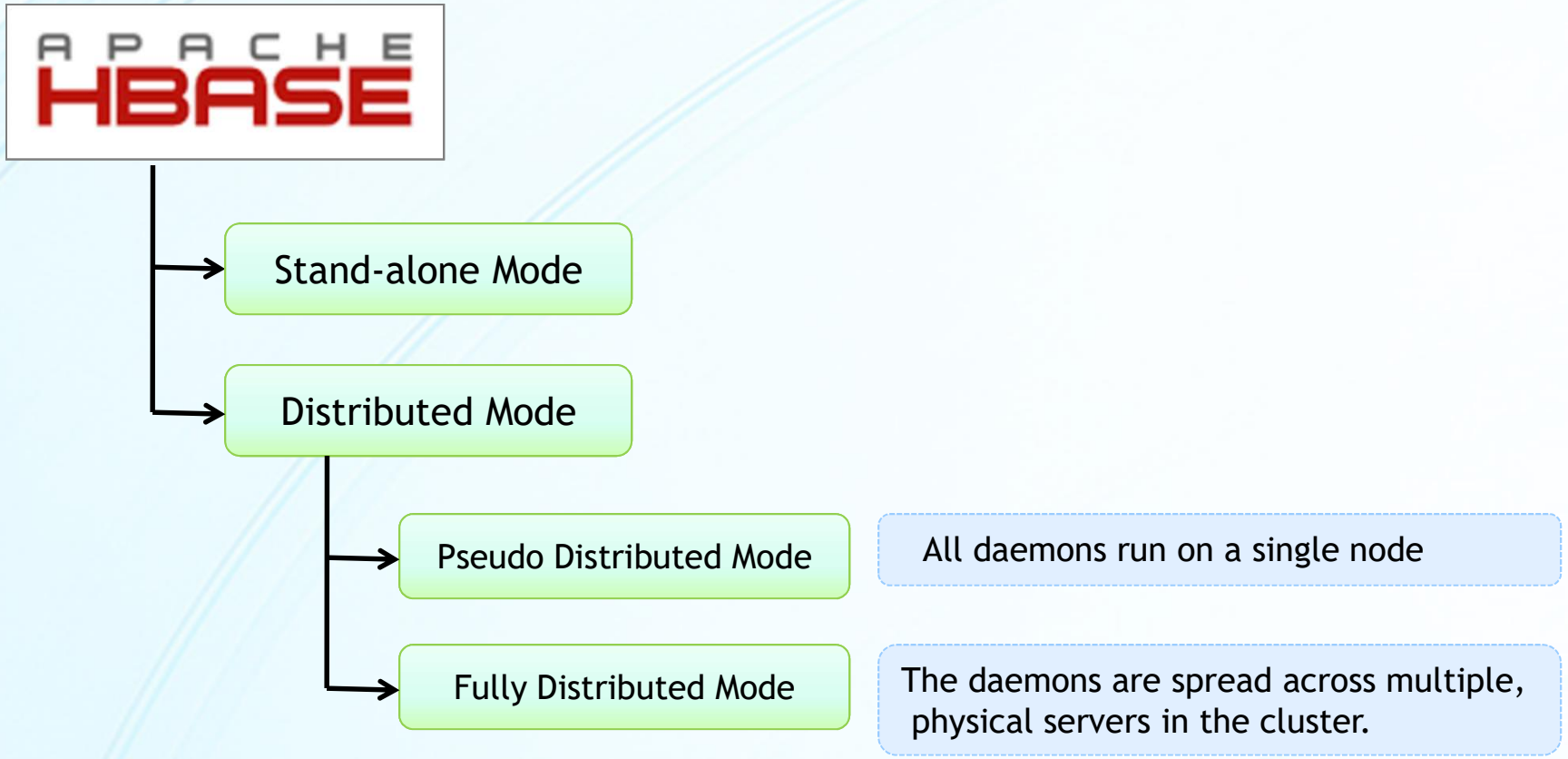
Messaging Infrastructure

HBase Run Modes

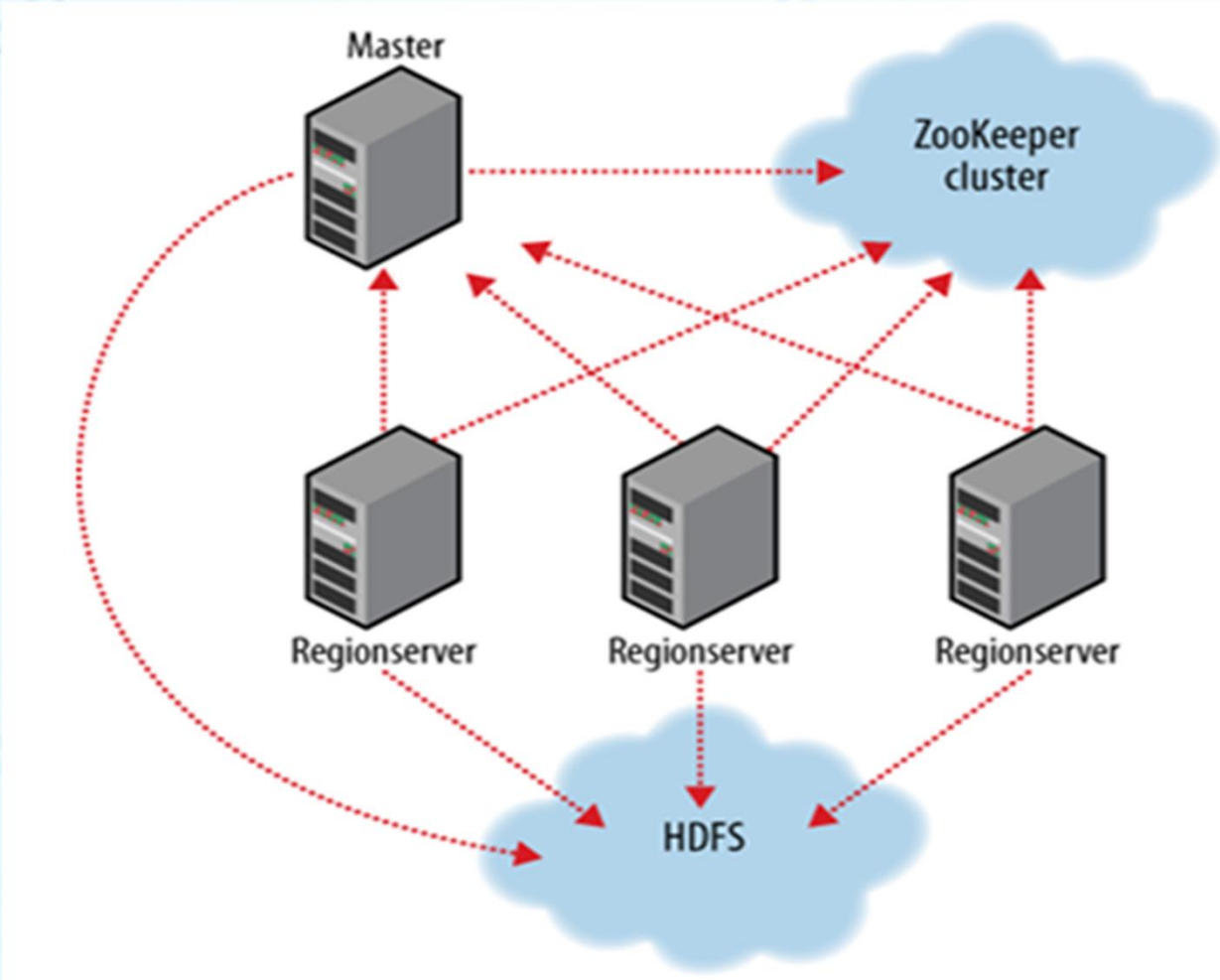
Stand-alone Mode

- **D**efault mode
- **D**oes not use HDFS — uses local file system
- **R**uns all HBase daemons and a local ZooKeeper in the same JVM process
- **Z**ooKeeper binds to a well-known port so that clients may talk to HBase.

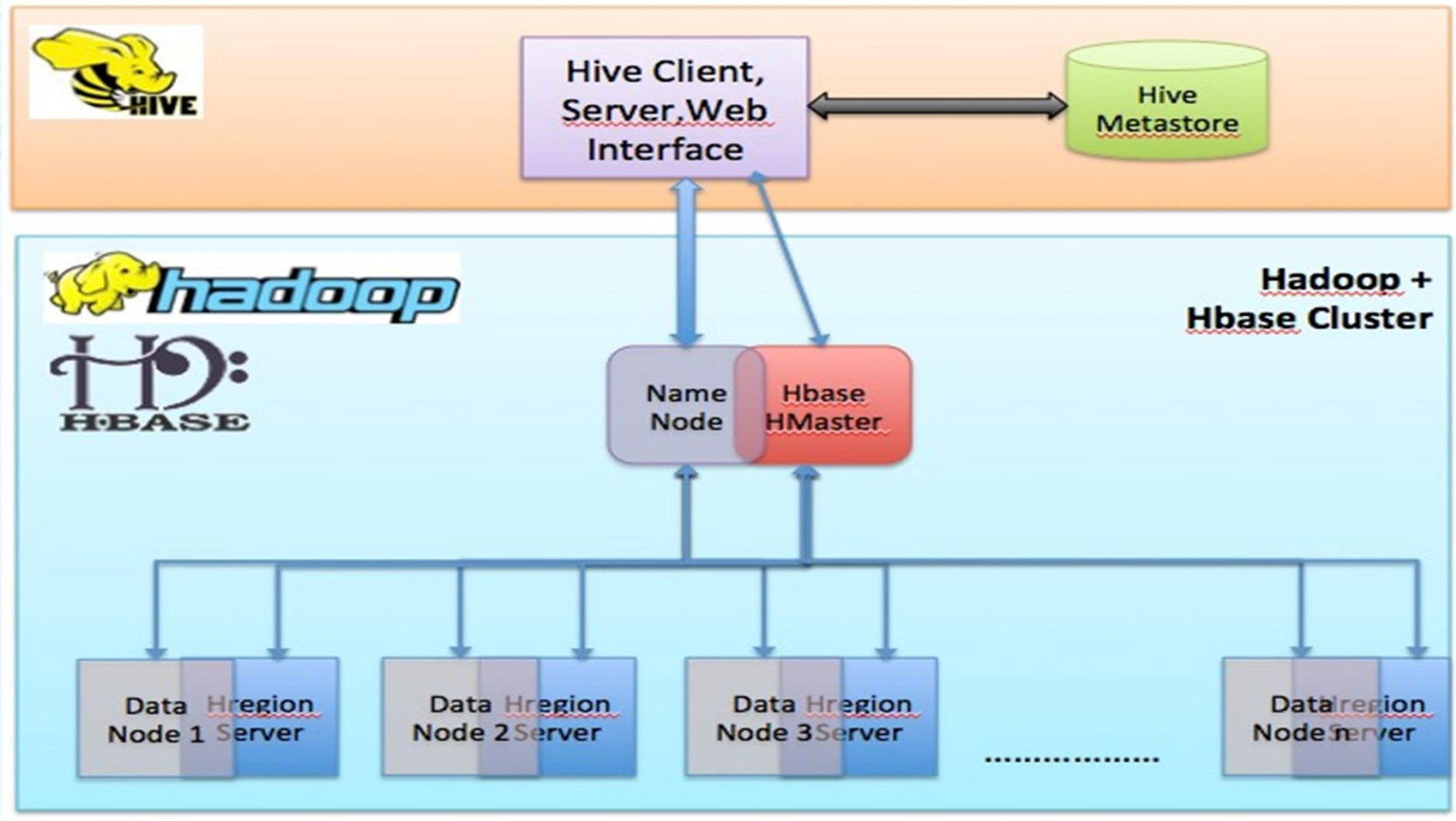
HBase Run Modes



HBase Architecture



HBase in Hadoop Ecosystem



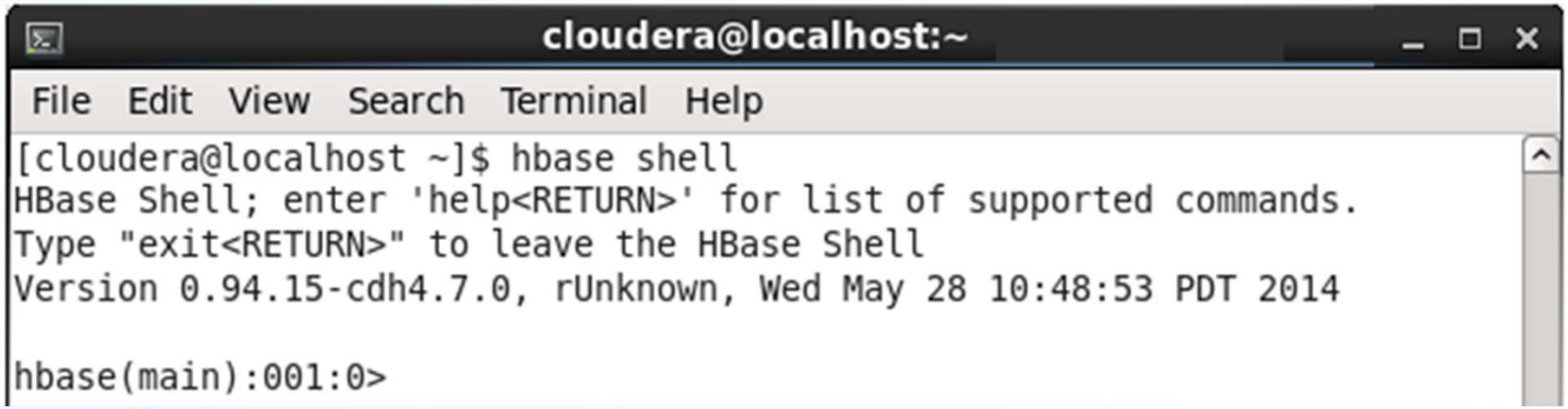
Installing & Running HBase

- HBase installs and runs on top of Hadoop. Hadoop needs to be installed and running before you install HBase.
- Once HBase is installed using either tar-balls or RPM packages, the main config file that has to be edited is **`hbase-site.xml`**
- The HBase HMaster process typically runs on the same machine where the Hadoop NameNode is installed
- The HBase HRegionServer processes run on each of the DataNodes along with the Hadoop DataNode and NodeManager processes

Invoking HBase Shell

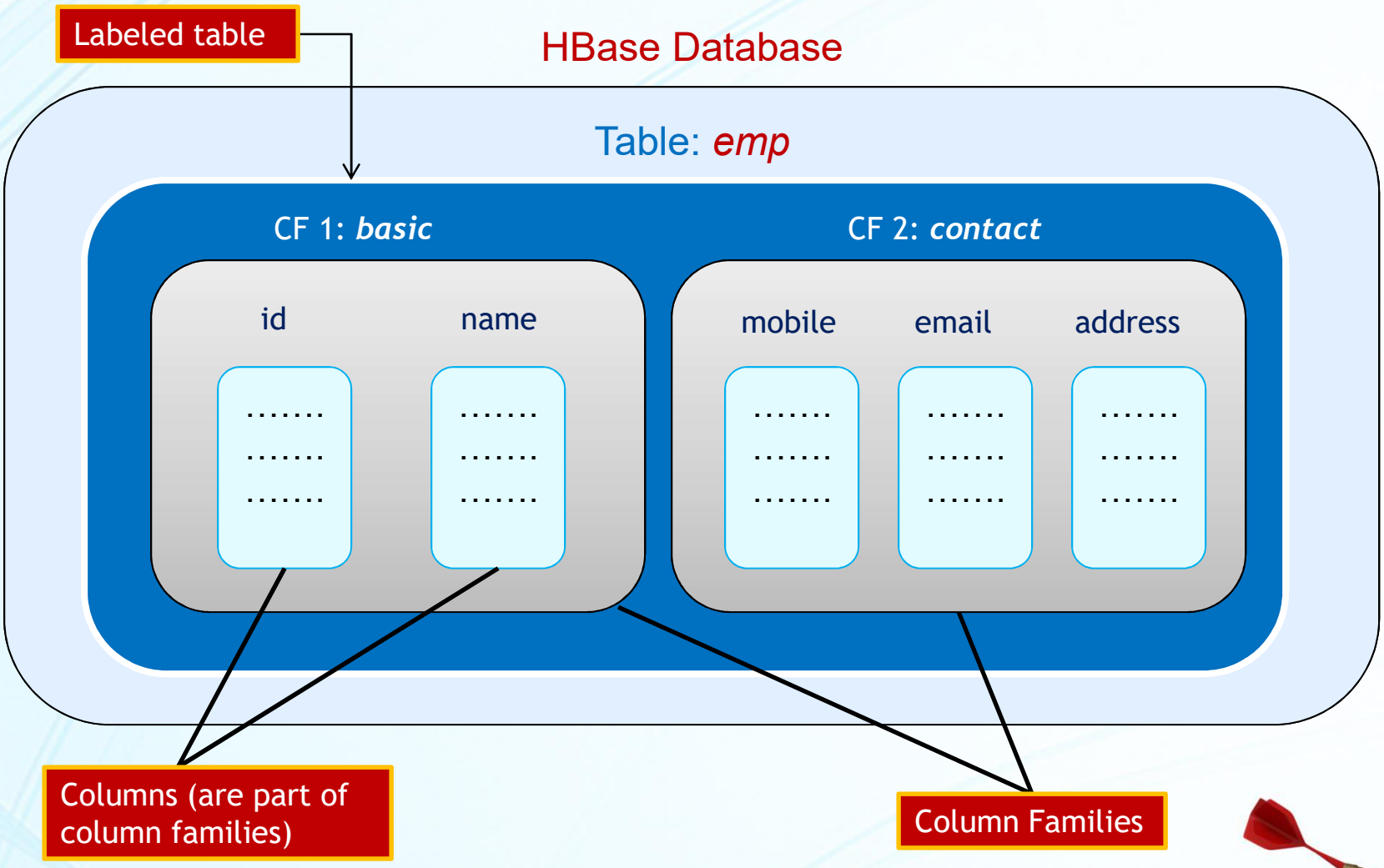
You can enter into HBase Shell using the following command:

```
$hbase shell
```



Let’s understand how data is organized in HBase

HBase Data Model



HBase Data Model

HBase Data Model

- Labeled Tables
- Column-Families
- Arbitrary number of Columns in a row
- Qualifiers (cf:cname)
- Versioned Data
- Unique Row-Ids
- Data as K-V pairs

HBase Data Model

In HBase, data is stored in **labeled tables**. A table definition consists of the table name (label) and a set of column-families.

A column-family may have *any number of columns*.

Columns are NOT part of table definition. Columns are assigned at the time of data insertion and *every row may not have the same number of columns*.

A table's column families must be specified up front as part of the table schema, but new column family members can be added on demand (using 'alter' command).

All column family members *have a common prefix*. For example, the columns *basic:id* and *basic:name* are both members of the *'basic'* column family. In this example, '*id*' and '*name*' are called *qualifiers*.

HBase Data Model

Table cells are versioned. By default, their version is a timestamp auto-assigned by HBase at the time of cell insertion.

Table data is stored as key-value pairs. Both the key and the value are byte-arrays.

Every row should contain a unique row-key. Data gets automatically sorted based on row-key.

HBase automatically adds a timestamp to track the versions of the data.

Let’s look at HBase Shell Commands

Create a HBase table

```
create 'emp', 'basic', 'contact', 'salary', 'skills'

// table-name: t1, family: f1, versions to mintain: 4
create 't1', {NAME => 'f1', VERSIONS => 4}
create 't1', {NAME => 'f1', VERSIONS => 4, TTL => 2592000}

// table-name: t1, families: f1, f2, f3
create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}

create 't1', 'f1', 'f2', 'f3'  // same as above
```

Alter a table

```
// alter or add a column-family
alter 't1', NAME => 'f1', VERSIONS => 5

// delete a column-family
alter 't1', NAME => 'f3', METHOD => 'delete'
```

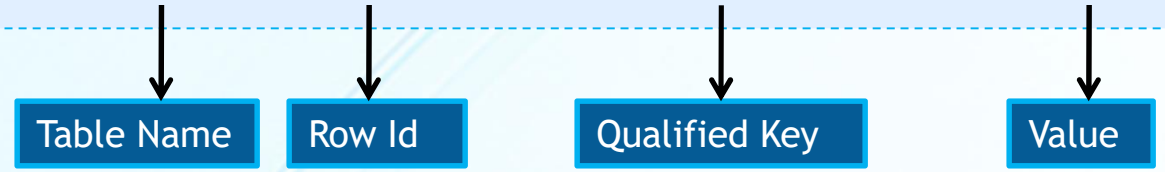
Drop a table

```
disable 'emp'
drop 'emp'

NOTE: to drop, you need to disable the table
```

Insert data in a Table

```
put 'emp', 'r001', 'basic:empid',    '100'
put 'emp', 'r001', 'basic:empname', 'Raju'
put 'emp', 'r001', 'basic:deptid',   '1'
put 'emp', 'r001', 'contact:email', 'raju@gmail.com'
put 'emp', 'r001', 'contact:phone', '9876543210'
```



Scan a table

Scan command scans the entire table and gets you the data of the entire table.

```
scan 'emp'
scan 'emp', {COLUMNS=>['basic:name', 'basic:id']}
scan 'emp', {COLUMNS => ['id', 'name'], LIMIT => 10, STARTROW => 'r0003'}
scan 'emp', {COLUMNS => 'id', TIMERANGE => [1303668804, 1303668904]}
```

```
hbase(main):038:0> scan 'empinfo'
ROW                          COLUMN+CELL
 100ABCD                     column=employee:firstName, timestamp=1545966262094, value=Veer4
 100_Veer                    column=dept:dname, timestamp=1545966108929, value=Operations
 100_Veer                    column=employee:DOB, timestamp=1545966081267, value=12-12-2012
 100_Veer                    column=employee:firstName, timestamp=1545965933625, value=Veer
 100_Veer                    column=employee:lastName, timestamp=1545966053288, value=Nagaraju
 100_seshu                   column=employee:hd, timestamp=1545966316687, value=12-12-2010
 100_seshu                   column=employee:lastName, timestamp=1545966331643, value=xxxxxxxx
3 row(s) in 0.0460 seconds
```

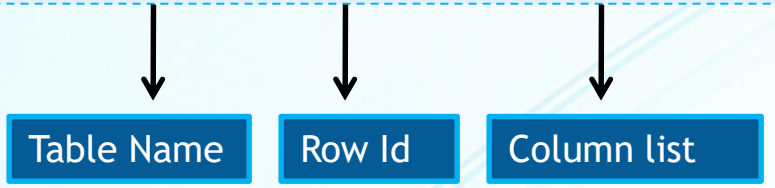
Query data from a table

```
get 't1', 'r1'
get 't1', 'r1', {TIMERANGE => [ts1, ts2]}
get 't1', 'r1', {COLUMN => 'cf1:c1'}
get 't1', 'r1', {COLUMN => ['cf1:c1', 'cf1:c2', 'cf1:c3']}
get 't1', 'r1', {COLUMN => 'cf1:c1', TIMESTAMP => ts1}
get 't1', 'r1', {COLUMN => 'cf1:c1', VERSIONS => 4}
get 't1', 'r1', 'cf1:c1'
get 't1', 'r1', ['cf1:c1', 'cf1:c2']
```

Table Name | Row Id | Column list | Column Options

Delete a cell

```
delete 'emp', 'r001', 'basic:name'
```



Describe the table

```
describe 'emp'
```

cloudera@localhost:~

File Edit View Search Terminal Help

```
hbase(main):006:0> describe 'empinfo'
DESCRIPTION                                                          ENABLED
 'empinfo', {NAME => 'address', DATA_BLOCK_ENCODING => 'N true
 ONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', V
 ERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '
 0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false',
 BLOCKSIZE => '65536', IN_MEMORY => 'false', ENCODE_ON_DI
 SK => 'true', BLOCKCACHE => 'true'}, {NAME => 'dept', DA
 TA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPL
 ICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'N
 ONE', MIN_VERSIONS => '0', TTL => '2147483647', KEEP_DEL
 ETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY =
 > 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true
 '}, {NAME => 'employee', DATA_BLOCK_ENCODING => 'NONE',
 BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSION
 S => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TT
 L => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKS
 IZE => '65536', IN_MEMORY => 'false', ENCODE_ON_DISK =>
 'true', BLOCKCACHE => 'true'}, {NAME => 'salgrade', DATA
 _BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLIC
 ATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NON
 E', MIN_VERSIONS => '0', TTL => '2147483647', KEEP_DELET
 ED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY =>
 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}
1 row(s) in 0.0800 seconds
```

Executing a HBase Script

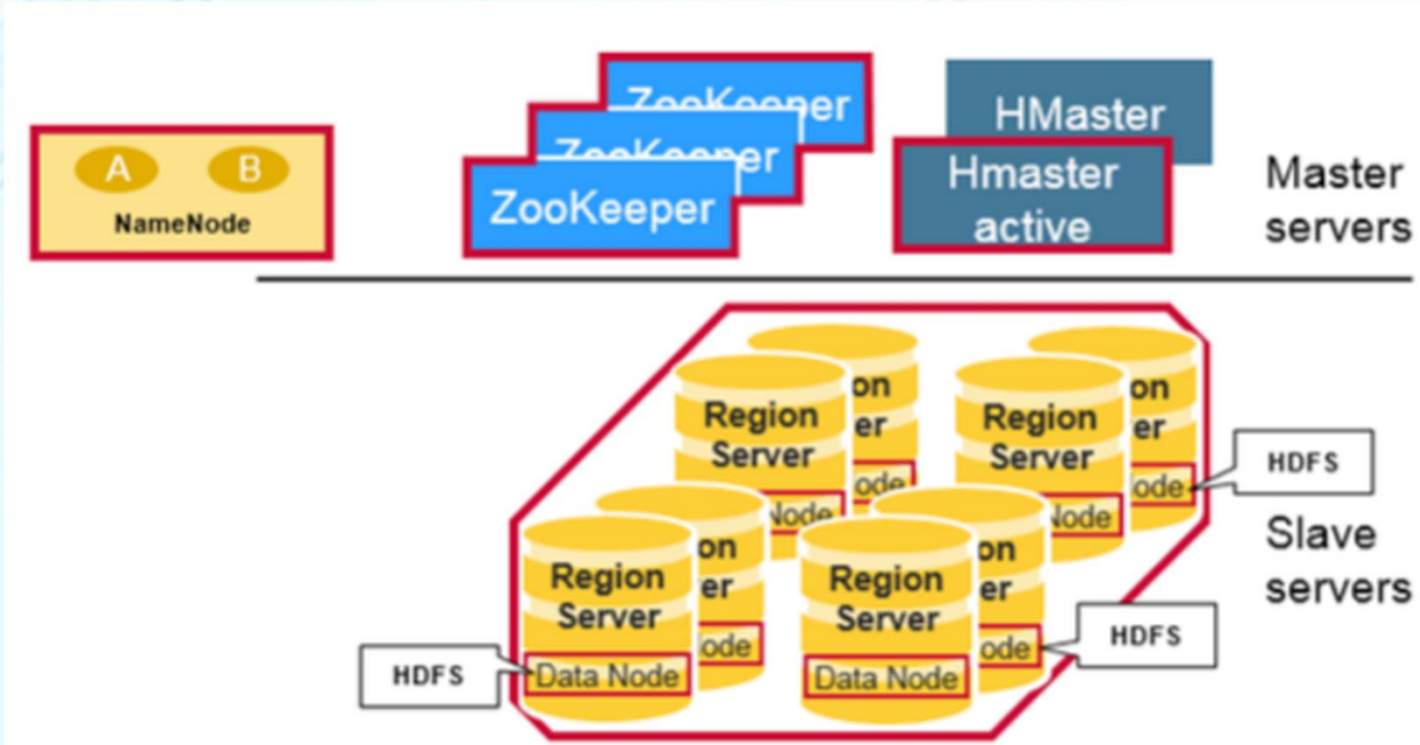
```
$hbase shell ./scriptfile.txt
```

The script file to be executed

- It is more common to execute script files than running ad-hoc commands from the shell.
- You can write all the HBase commands in a single file and execute all the commands in that script file using the above command.

HBase Architecture

HBase Architecture



HBase Daemons: **HMaster** (master daemon) & **RegionServer** (slave daemon)

HBase Components

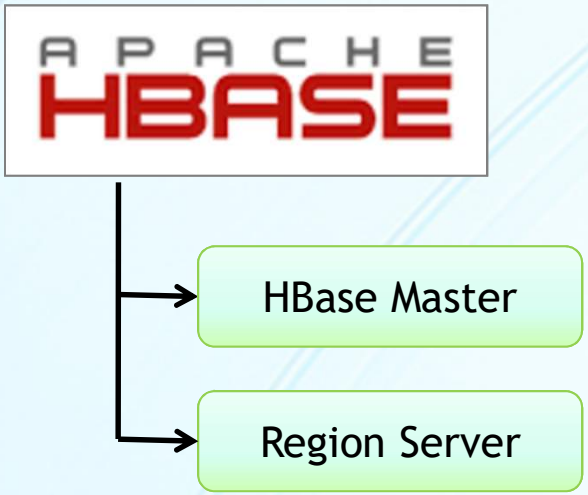
HBase follows master-slave architecture. HBase is made up of an HBase 'Master' node orchestrating a cluster of one or more 'RegionServer' workers.

HBase Master

- Responsible for bootstrapping a virgin install
- Assigns regions to registered region-servers
- Responsible for recovering RegionServer failures
- Is lightly loaded

HBase Components

HBase follows master-slave architecture. HBase is made up of an HBase 'Master' node orchestrating a cluster of one or more 'RegionServer' workers.



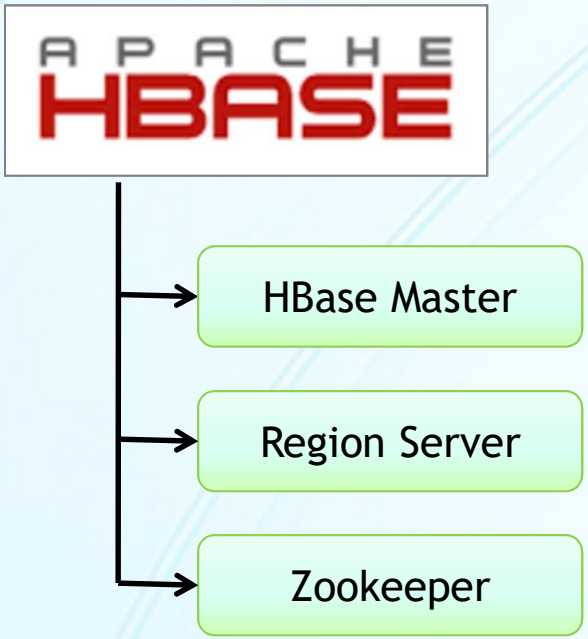
Carry zero or more regions

Field client read/write requests

Manage region splits, informing the HBase master about the new regions

HBase Components

HBase follows master-slave architecture. HBase is made up of an HBase 'Master' node orchestrating a cluster of one or more 'RegionServer' workers.

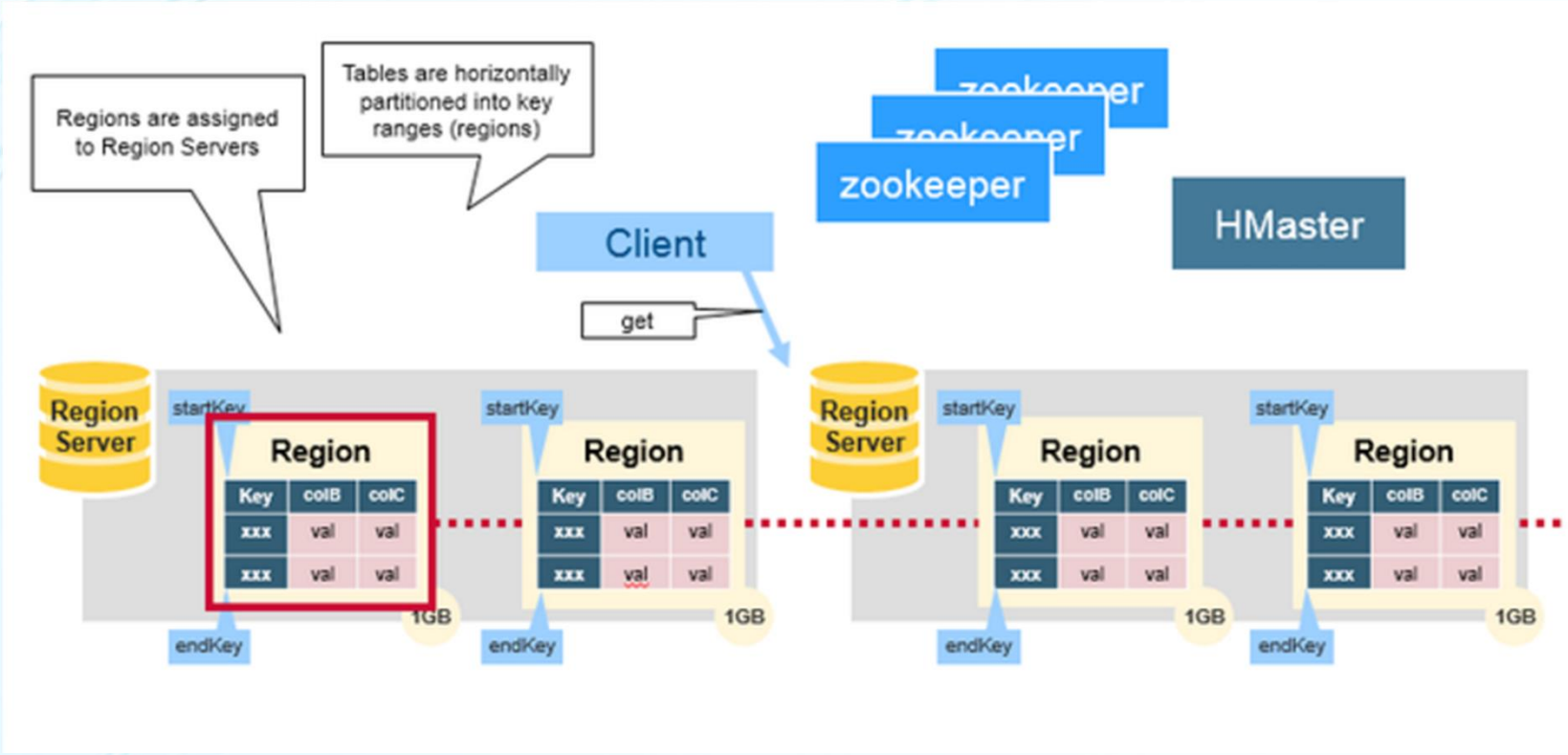


Authority on cluster state

Hosts vitals like location of the hbase:meta catalog table and the address of the current cluster master.

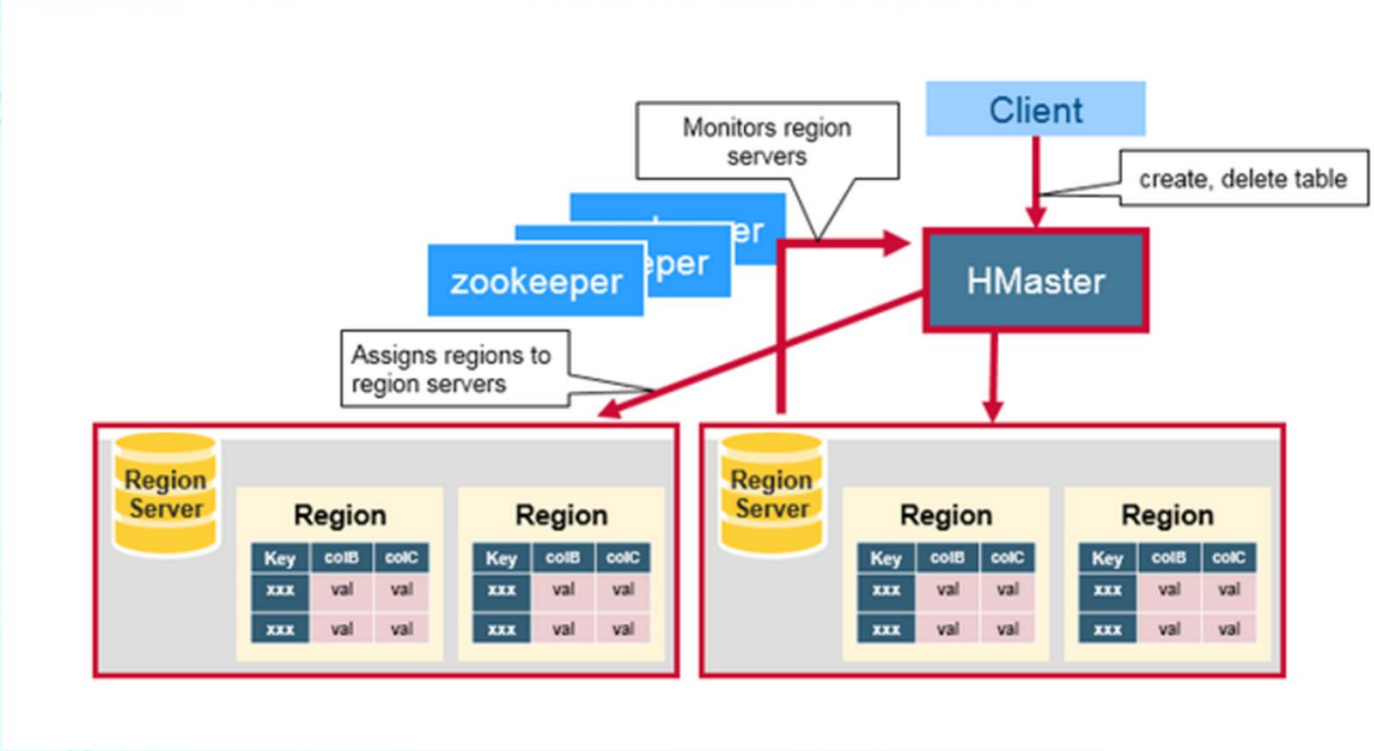
Assignment of regions is mediated via ZooKeeper

HBase Architecture contd..



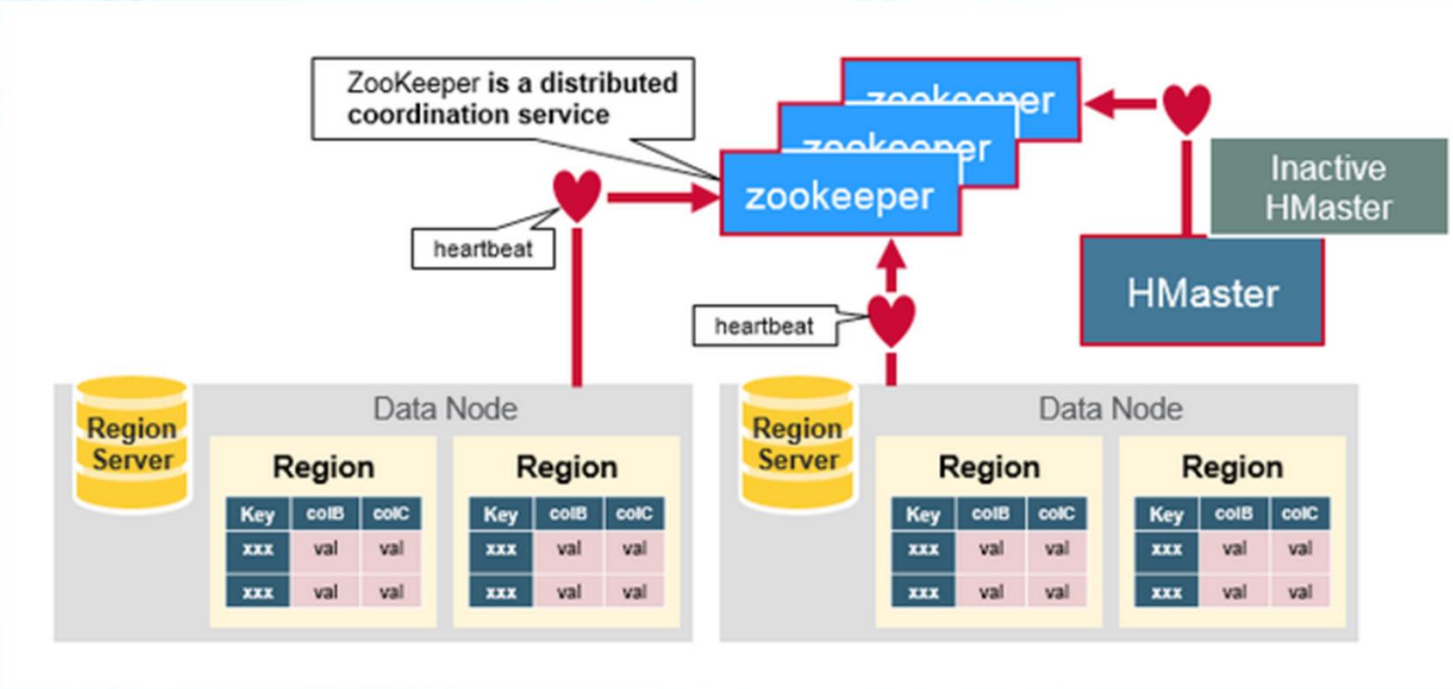
- Tables are horizontally partitioned into key regions (regions)
- Regions are assigned to RegionServers

HBase Architecture contd..



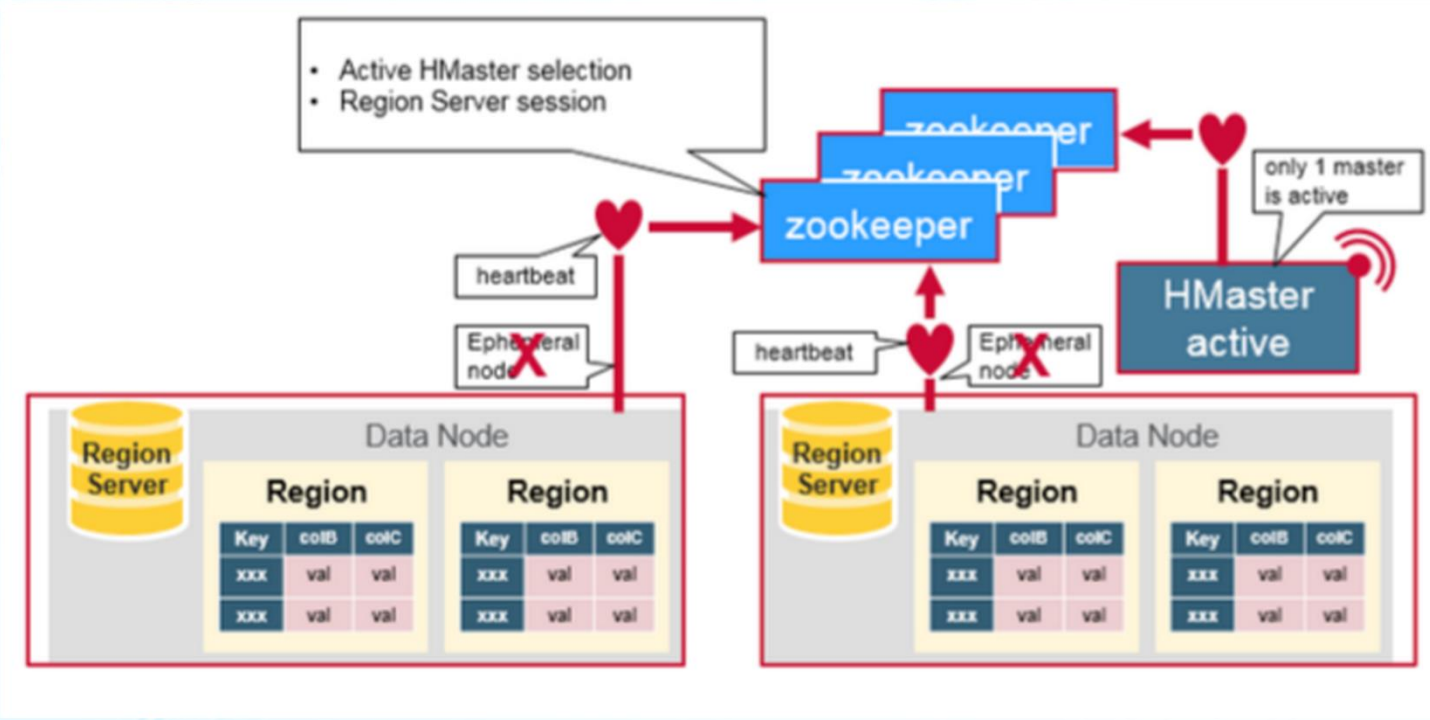
- HMaster assigns regions to RegionServers
- HMaster monitors RegionServers

HBase Architecture contd..



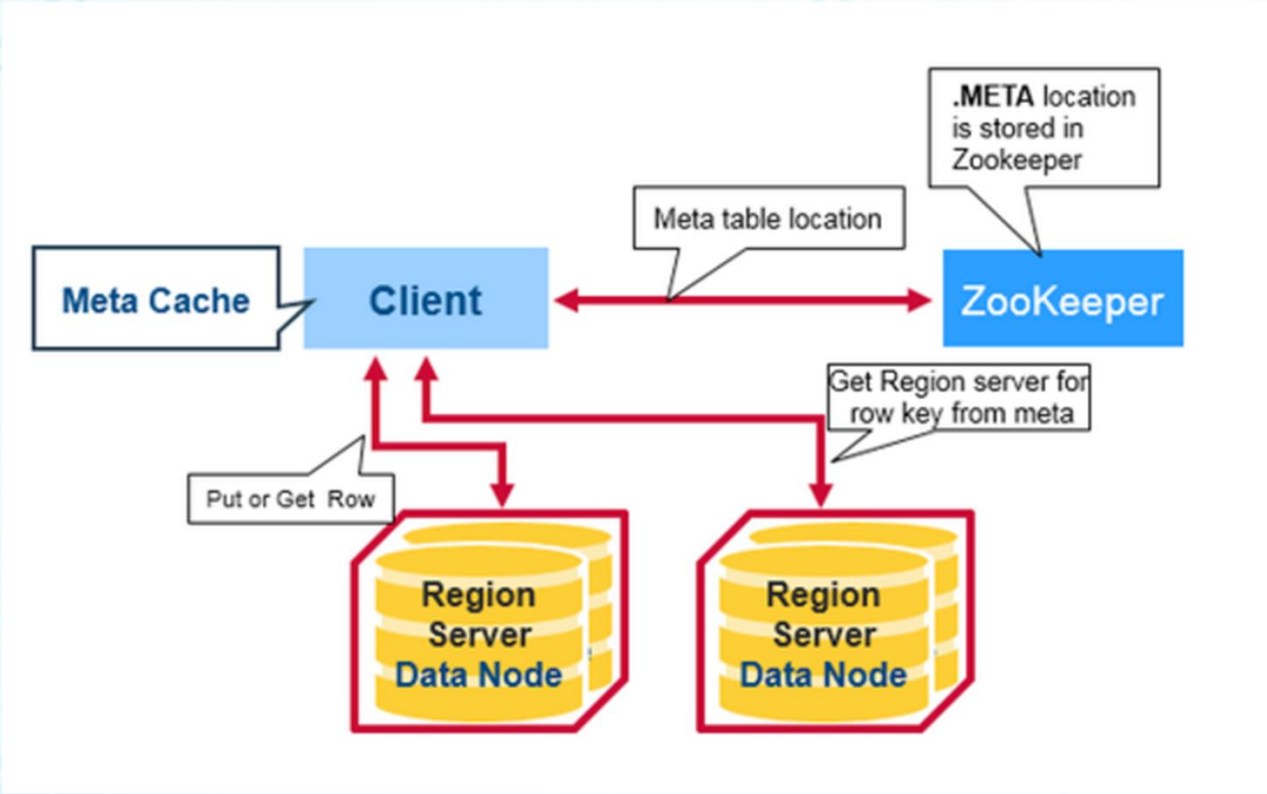
- Zookeeper acts as distributed coordination service
- HMaster and RegionServers send heartbeats to ZooKeeper

HBase Architecture contd..



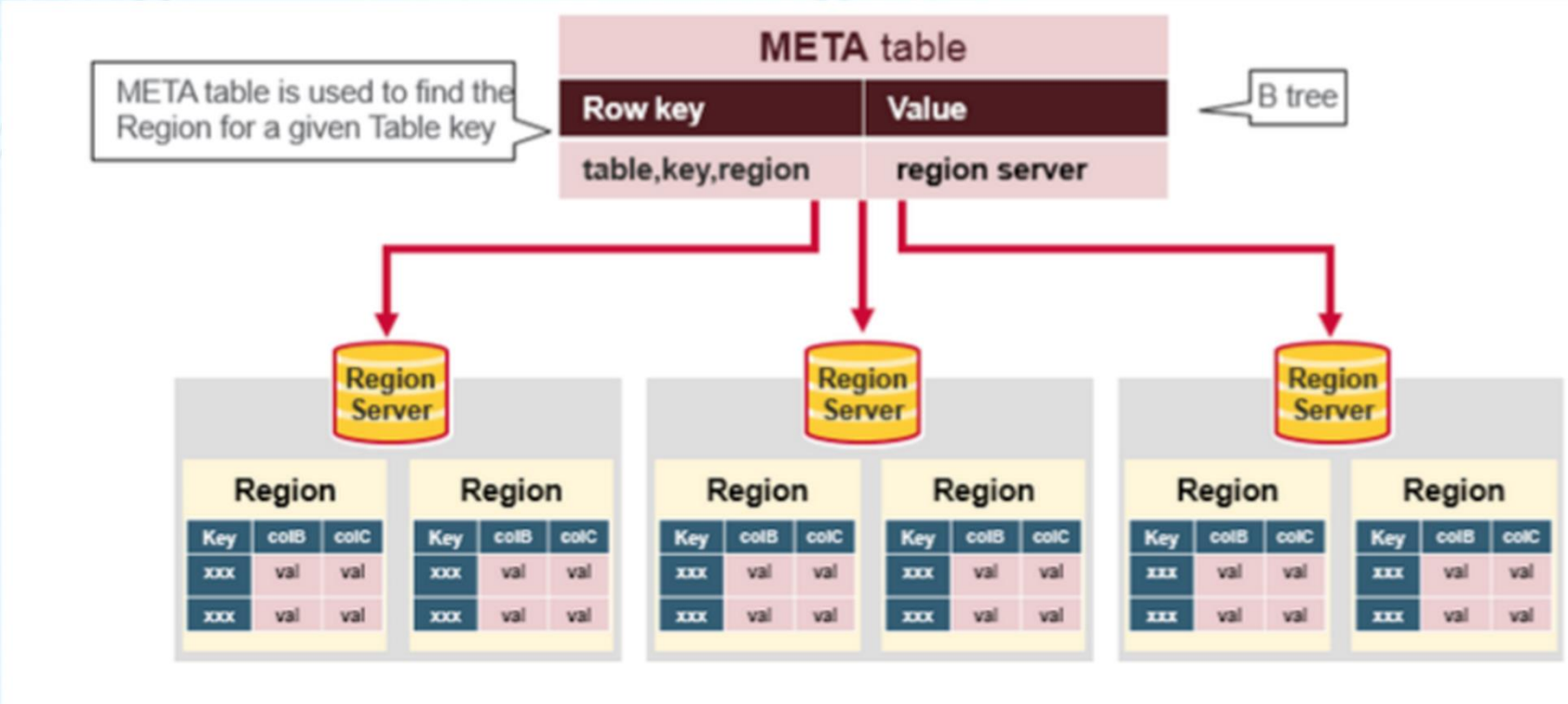
- Zookeeper manages active HMaster selection and monitors RegionServer session.

HBase Architecture contd..



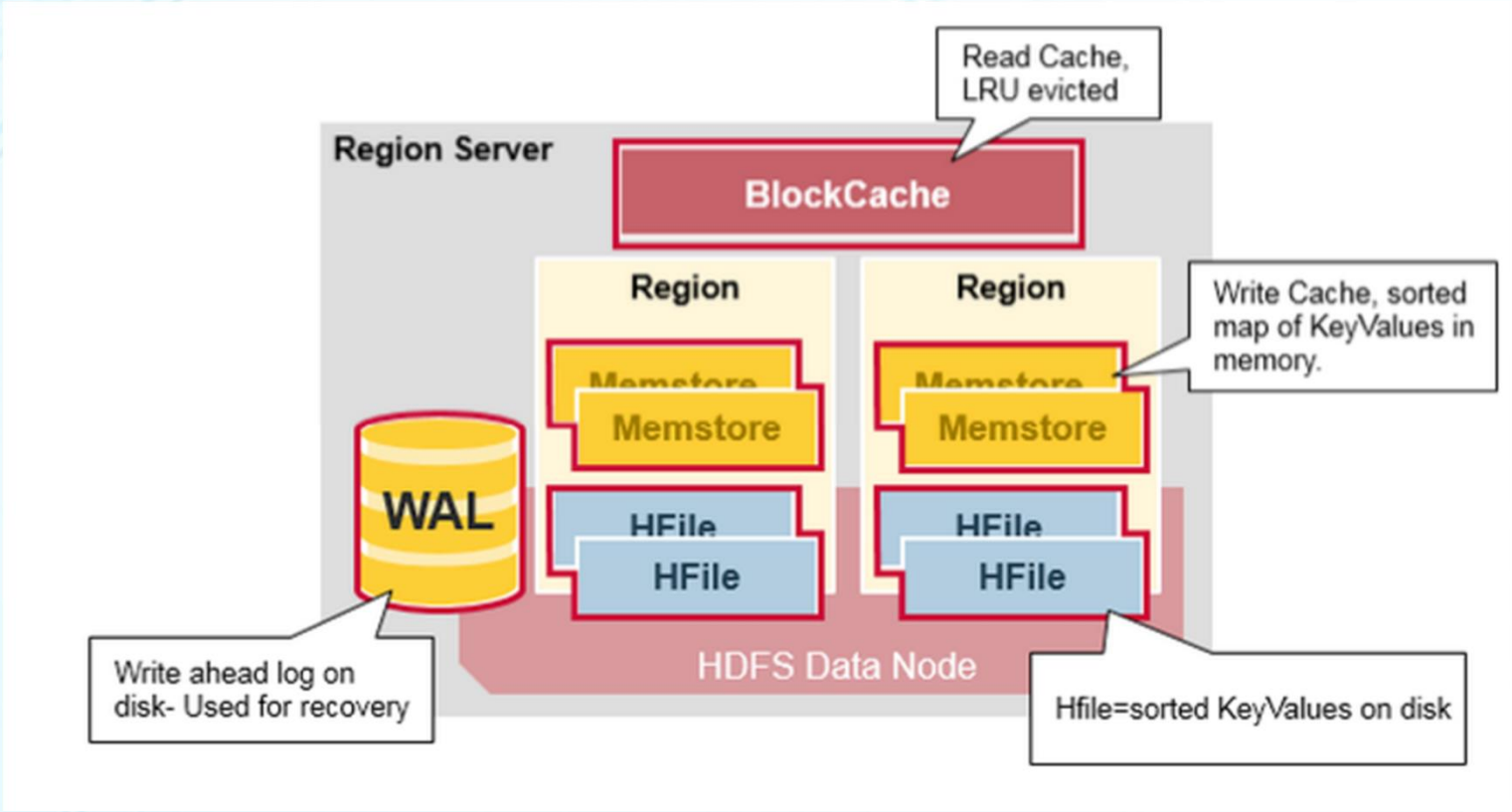
Client receives meta info from ZK to decide which RegSrvr to read from or write to.

HBase Architecture contd..



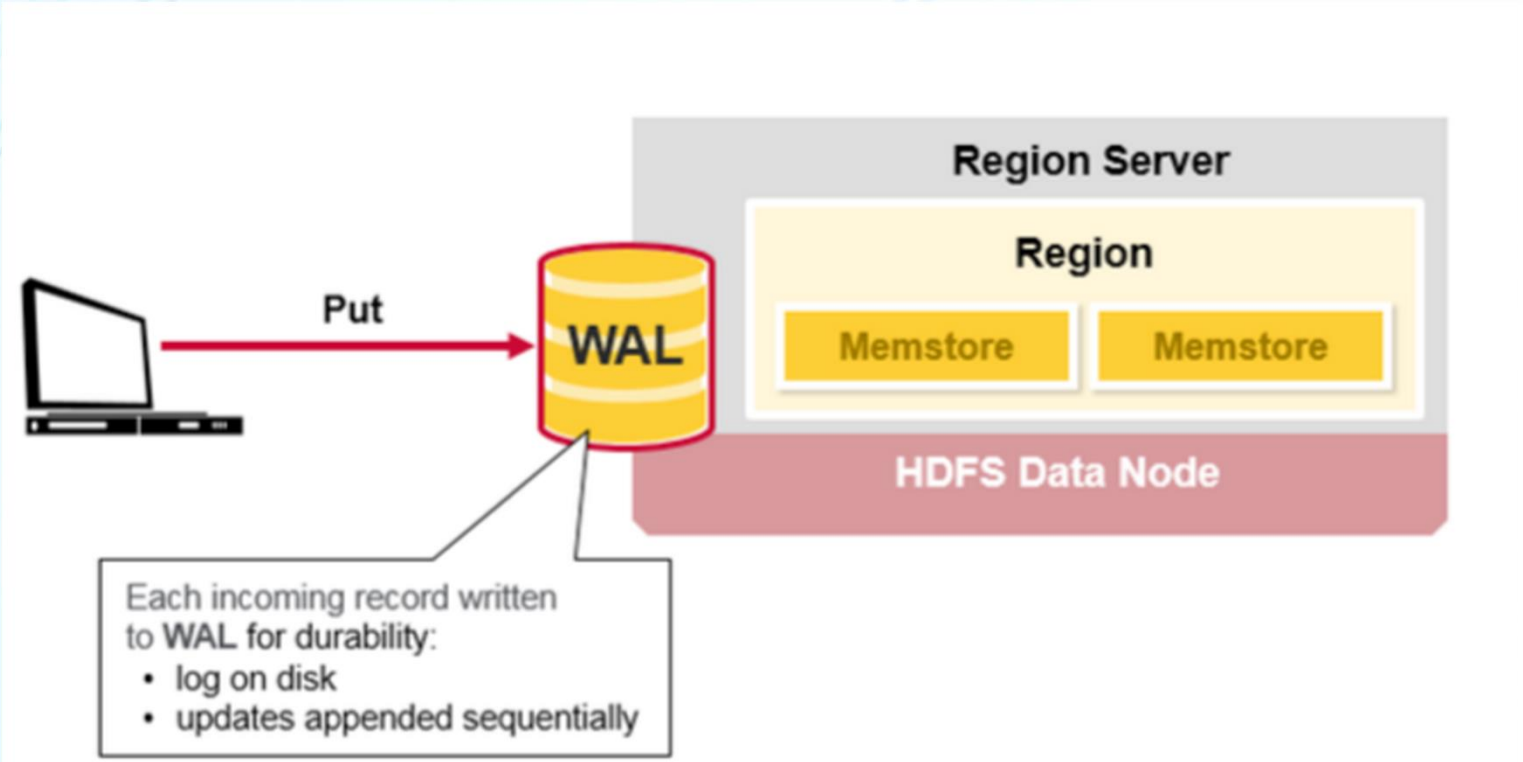
Meta table is used to find the Region for a given table key.

HBase Architecture contd..



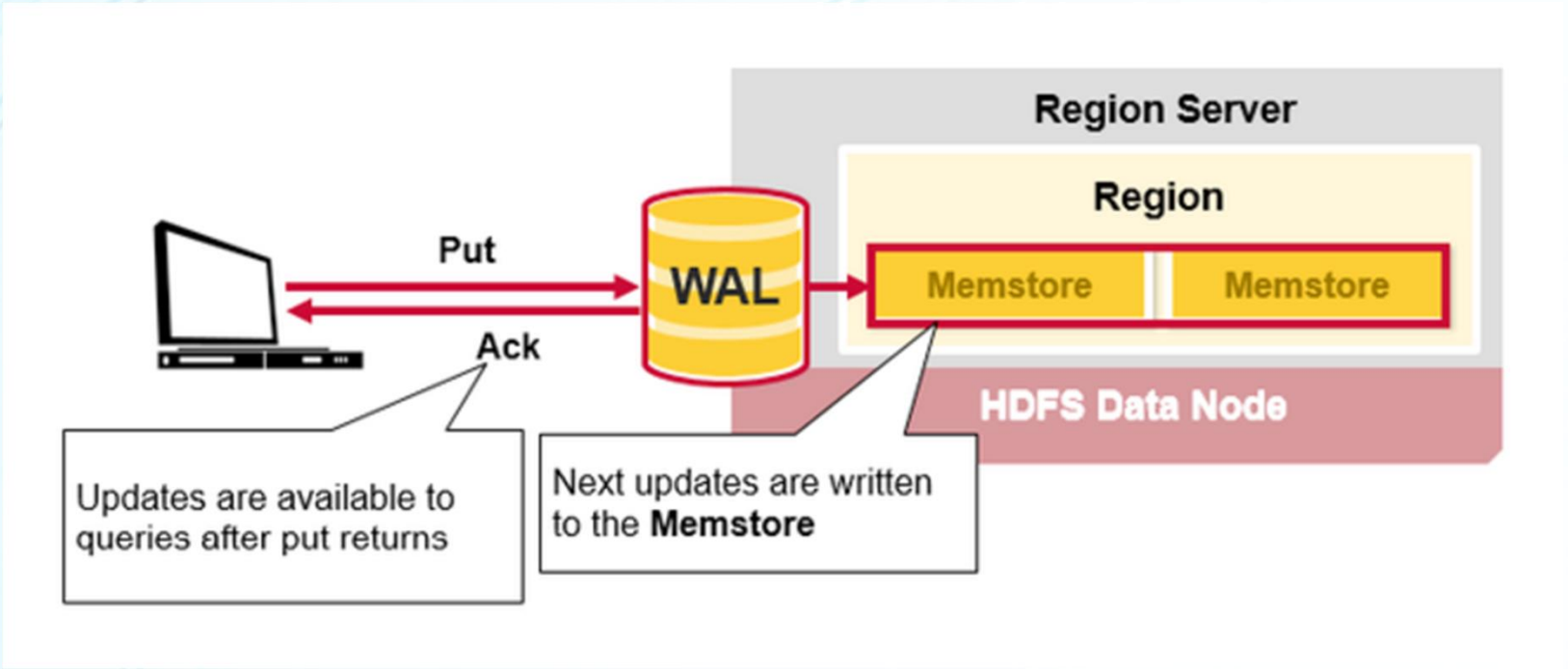
Writes arriving at a RegSrvr are first appended to a 'commit log' (WAL) and then added to an in-memory 'MemStore'. There is one MemStore for each column-family. That's why we say, column-family data gets sorted and stored at one place.

HBase Architecture contd..

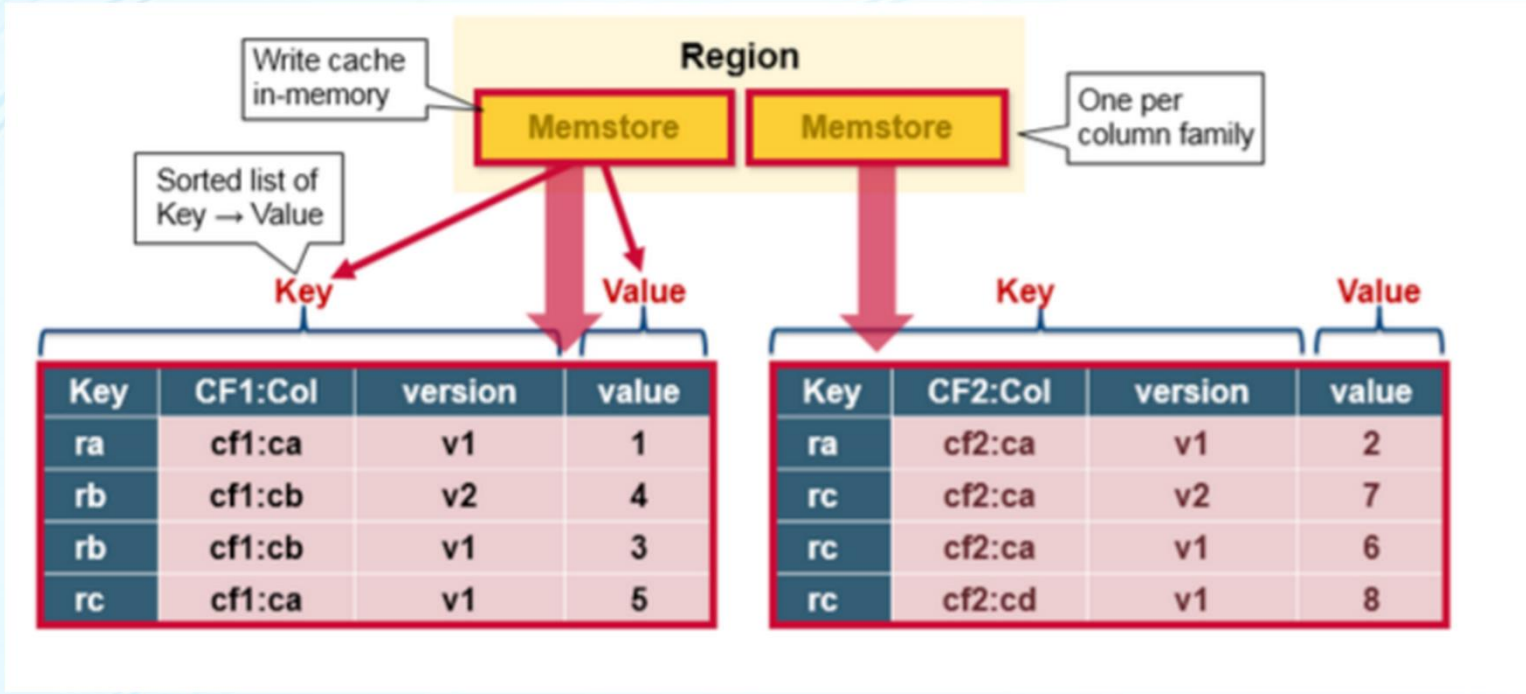


Each incoming record is written to WAL. WAL is stored on HDFS and updates are appended sequentially.

HBase Architecture contd..



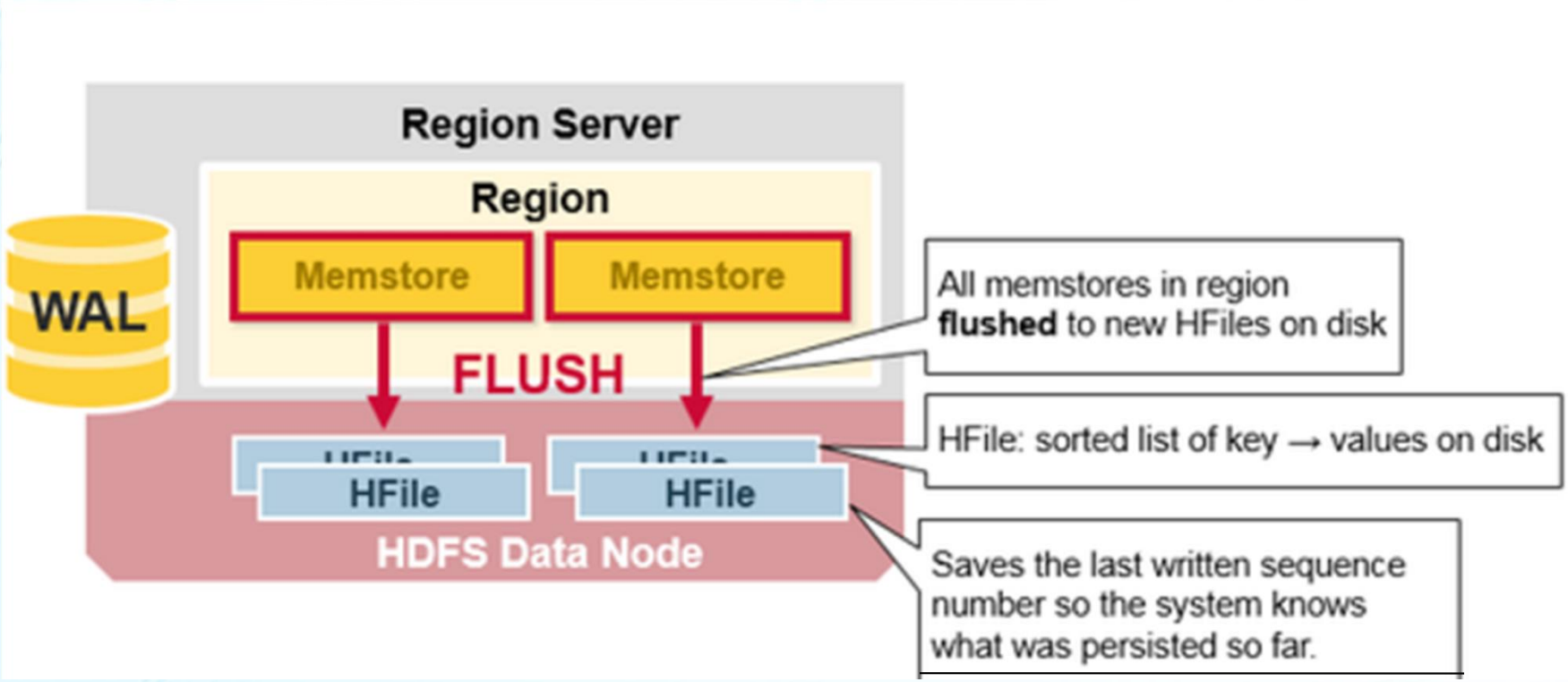
HBase Architecture contd..



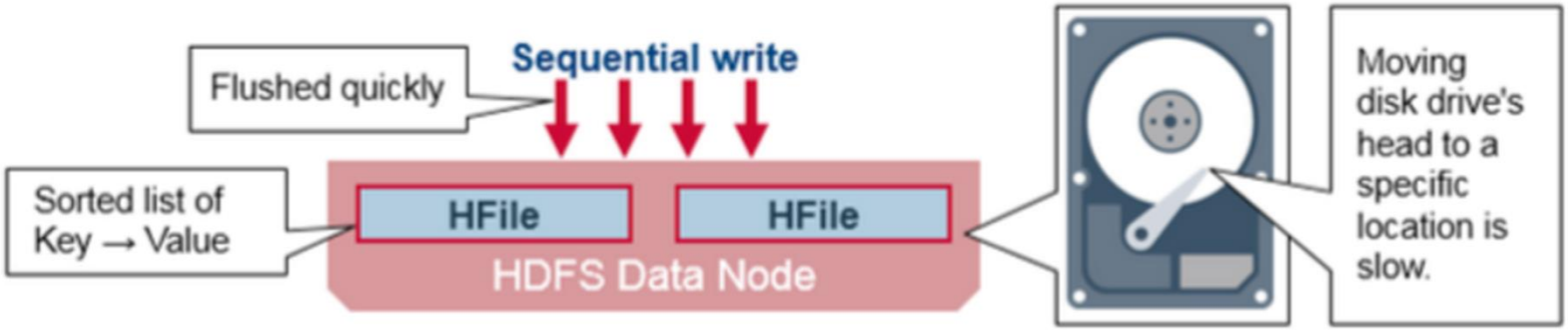
Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

HBase Architecture contd..



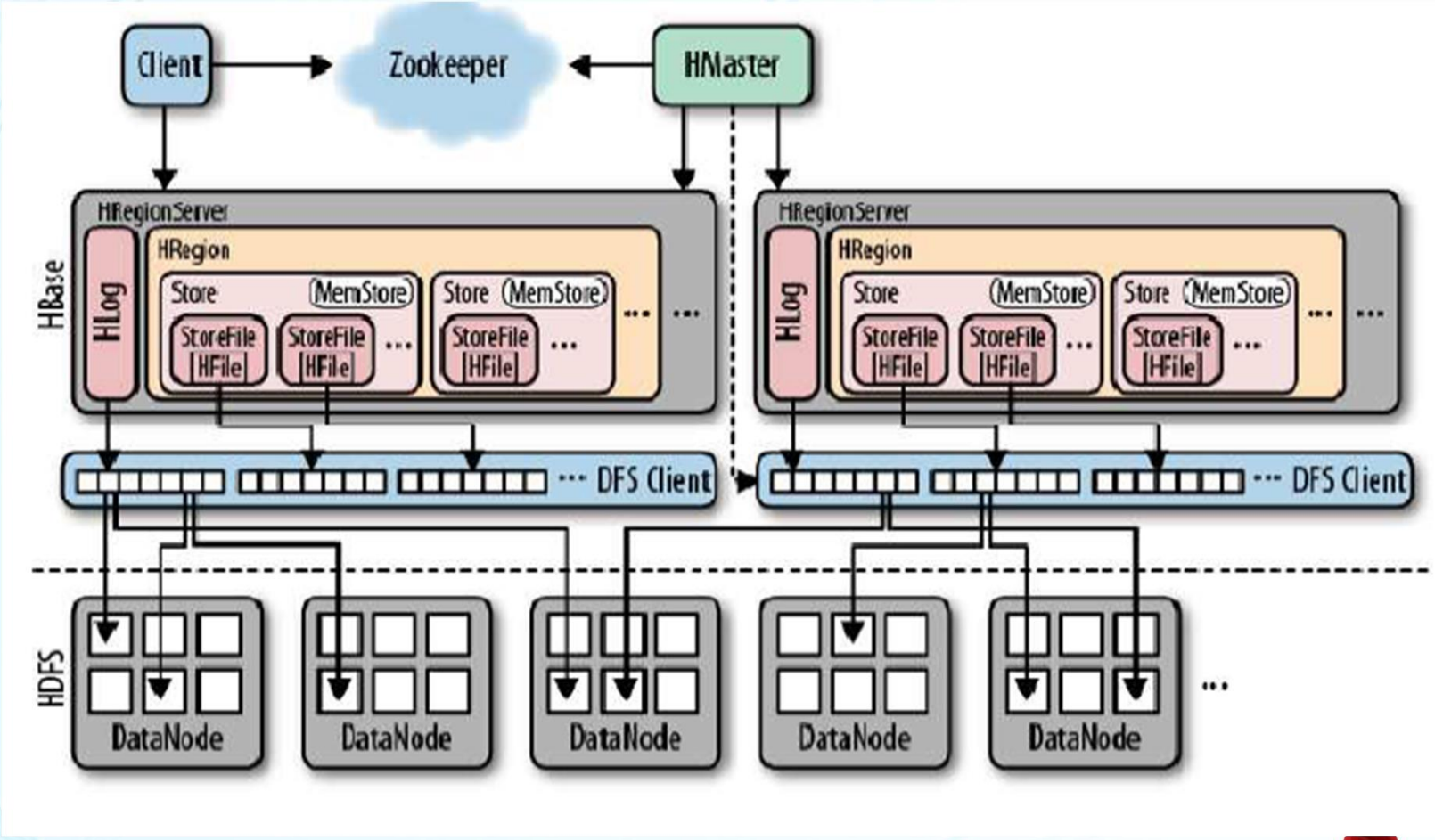
HBase Architecture contd..



Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

HBase Architecture contd..



Compaction

- The store files are monitored by a background thread to keep them under control.
- The flushes of MemStores (as HFiles) slowly build up an increasing number of on-disk files.
- If there are enough of them, the compaction process will combine them to a few, larger files.
- This goes on until the largest of these files exceeds the configured maximum store file size and triggers a region split.

THANK YOU